The VU68K Single-Board Computer

A 68000-based system for only $200

by Edward M. Carter and A. B. Bonds

As more and more 16-bit microprocessors appear on the market, the question of which one to base a development or hobbyist system on becomes more difficult. Ideally, most hobbyists and students would base their decision on thorough hands-on experience, but getting hold of inexpensive trainers is almost impossible. There is relief, however—the VU68K, a complete 68000-based single-board computer that can be constructed for under $200. The VU68K is not a toy but a powerful dual-ported computing engine that is limited only by its small memory, which can easily be expanded.

Hardware Description

We set out to design a simple and low-cost 16-bit microprocessor system that would offer maximum utility in both tutorial and development applications. Of the 16-bit processors that we examined, our overwhelming choice was the 68000, which offers both simplicity and substantial processing power. This power is evident in its comprehensive instruction set, which supports two processing modes and a powerful interrupt-processing facility (also called exception processing). A secondary, yet significant, benefit is the 68000's sales leadership. The VU68K system thus provides the user with marketable and timely knowledge. Additional reasons for selecting the 68000 include proven reliability and a commitment on the part of Motorola to maintain compatibility between its new chips and the 68000.

The 68000 uses "words" (the name for the instructions or data that the microprocessor operates on) of 16 bits or 2 bytes in "width." (Recall that an 8-bit microprocessor uses words of 8 bits in width.) We decided to include 2K words of ROM (read-only memory) with the VU68K for a monitor, and 2K words of RAM (random-access read/write memory) for developing and testing user programs. This gives the VU68K a total memory of 4K words in "length" and 16 bits in "width." In addition, we added two RS-232C-compatible serial ports for connection to a modem, a printer, or what have you.

We were able to include these features at very low cost. The parts list in table 1 shows prices from a major retail supplier of electronics parts. All of the parts are readily available and can be obtained from many sources. The prices shown are from a single catalog—a little bargain hunting would probably yield a much less expensive system. Note that the total cost of $190.22 is for a complete system built from scratch. It includes the costs of resistors, capacitors, wire, circuit board, etc. The cost of upgrading from an 8-bit computer to a VU68K configuration will be much less, assuming that the memories, circuit board, crystals, etc., from the old system can be retained.

Photo 1 and the schematic in figure 1 show how simple our design is. Only 15 ICs (integrated circuits) are required for the entire VU68K system. The functions of these 15 ICs can be divided into six categories: asynchronous bus operation, synchronous bus operation, interrupt handling, address decoding, communications interface, and miscellaneous support.

Asynchronous Bus Operation

The 68000 (IC2) is designed to communicate asynchronously on the system bus; that is, without a timing signal and with any amount of time between data bytes or words. To indicate when data has been received or sent, the device with which the 68000 is communicating sends an

acknowledgment signal to the 68000. This way, the device and the 68000 can operate at different rates and still communicate with each other, because one waits for the other to finish reading or writing. (See the *Motorola MC68000 16-Bit Microprocessor User's Manual* for the details on 68000 asynchronous bus communication.)

In the VU68K, only the RAM (IC9, IC10) and ROM (IC7, IC8) communicate asynchronously on the data bus. Since the memories don't themselves have an acknowledgment signal, we synthesized one with a synchronous 4-bit counter (IC4) driven by a simple oscillator circuit and 5-MHz crystal. When the address strobe signal, $\overline{AS}$, goes low to signal a valid address on the address bus, an initial count of 1100 is loaded into the counter. After four clock ticks, the high-order output bit, connected to the acknowledge pin $\overline{DTACK}$ (data transfer acknowledge) goes low. The transition signals that the data requested by the processor is available

Text continued on page 411

IC	Part Number	Description	Cost ($)
1	7400	Quad 2-input NAND Gate	.19
2	68000G8	MPU 16-bit (8 MHz)	69.95
3	7404	Hex Inverter	.25
4	74161	Synchronous 4-bit Counter	.69
5	74154	4-to-16 Decoder	1.25
6	7432	Quad 2-input OR Gate	.29
7	2716	16K EPROM 450ns	4.95
8	2716	16K EPROM 450ns	4.95
9	6116-P4	Static RAM 200ns 16K CMOS	6.95
10	6116-P4	Static RAM 200ns 16K CMOS	6.95
11	6850	Async. Comm. Int. Adapter	4.95
12	6850	Async. Comm. Int. Adapter	4.95
13	14411	Bit-rate Freq. Generator	11.95
14	1488	Quad Line Driver	.69
15	1489	Quad Line Receiver	.69
—	8800V	Vector Board	24.95
—	7812	12-volt Positive Regulator	.79
—	7912	12-volt Negative Regulator	.89
—	—	5 MHz Crystal	2.95
—	—	1.8432 MHz Crystal	4.95
—	—	Capacitors, Resistors, Wire	4.25
—	—	Pushbutton Switch	1.95
—	—	8-position DIP Switch	1.49
—	—	14-pin Wire-wrap Socket	5 @ .45
—	—	16-pin Wire-wrap Socket	2 @ .69
—	—	24-pin Wire-wrap Socket	8 @ 1.29
—	—	64-pin Wire-wrap Socket	14.40

Table 1: *Parts list.*

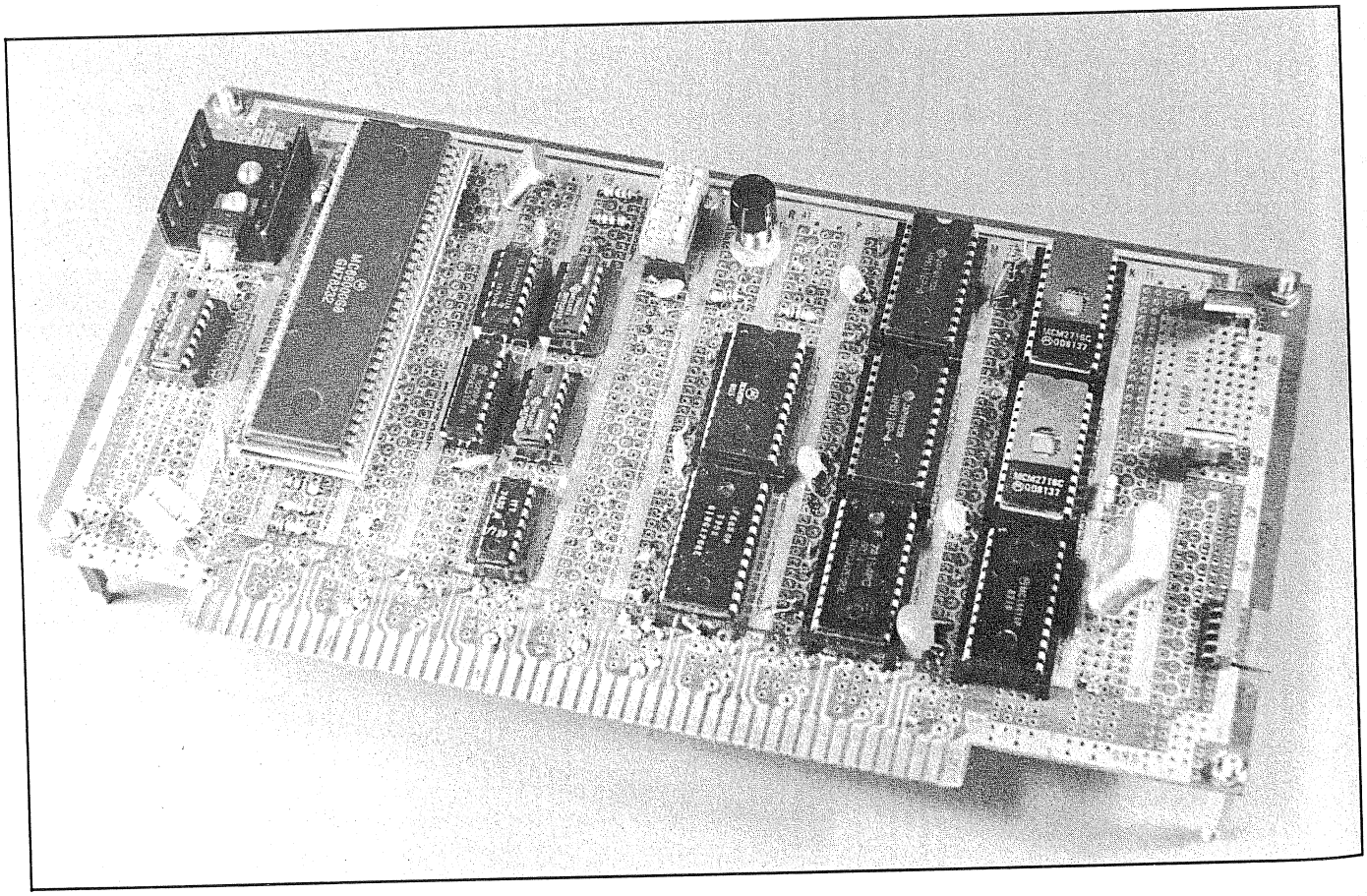


Photo 1: *The VU68K board.*

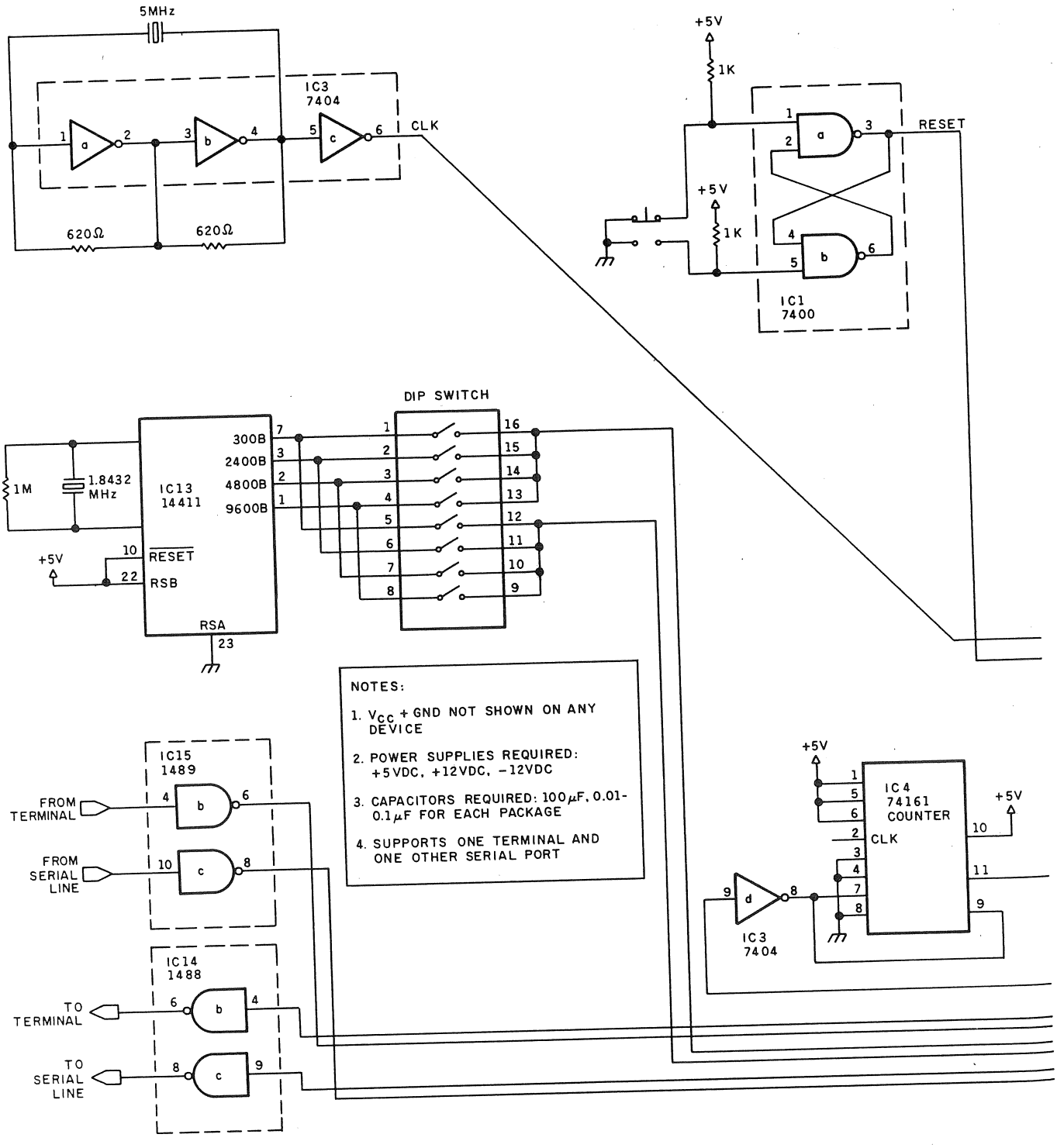
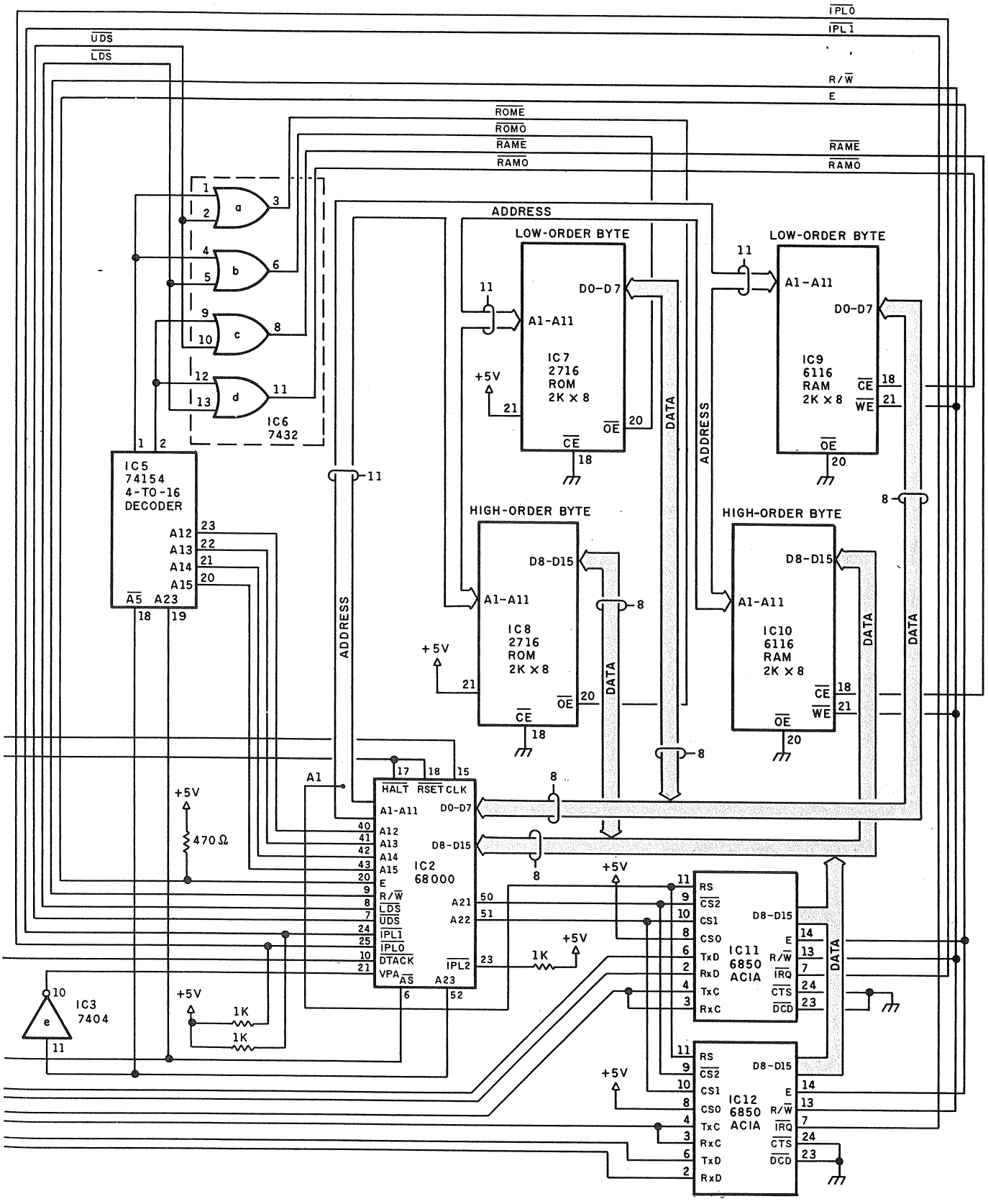


Figure 1: *The VU68K schematic.*


UDS
LDS
IPL0
IPL1
R/W
E
ROME
ROMO
RAME
RAMO
ADDRESS
LOW-ORDER BYTE
HIGH-ORDER BYTE
IC5
74154
4-TO-16
DECODER
IC6
7432
IC7
2716
ROM
2K × 8
IC8
2716
ROM
2K × 8
IC9
6116
RAM
2K × 8
IC10
6116
RAM
2K × 8
IC2
68000
IC3
7404
IC11
6850
ACIA
IC12
6850
ACIA
DATA
A1
+5V
470 Ω
1K
HALT
RSET
CLK
DTACK
VPA
AS
A1-A11
D0-D7
D8-D15
A12
A13
A14
A15
A21
A22
A23
IPL2
RS
CS2
CS1
CS0
TxD
RxD
TxC
RxC
IRQ
CTS
DCD

Text continued from page 404:

on the bus. With a clock speed of 5 MHz, this provides 800 nanoseconds (ns) for the transfer to occur.

Synchronous Bus Operation

In the design of the 68000, Motorola recognized that a large number of 8-bit synchronous peripherals are available and familiar to many people. These devices depend on the timing signals generated by the 6800 family of processors for data communication. To provide compatibility, the 68000 can also generate a synchronous timing signal.

When the $\overline{AS}$ line is asserted (low), the synchronous peripherals must pull the $\overline{VPA}$ (valid peripheral address) processor pin low to generate the synchronous timing signals $\overline{VMA}$ (valid memory address) and E (enable). In the VU68K, the only set of devices that uses synchronous transfers is the Asynchronous Communications Interface Adapters or ACIAs (IC11, IC12), which drive the two serial ports (note that the term asynchronous here refers to the RS-232C standard and not the bus protocol). These two ports may be accessed at addresses $A00000–$A00002 and $C00000–$C00002. (For the remainder of this article, the prefix "$" will denote a hexadecimal address.) Address line A23 is tied to the processor's $\overline{VPA}$ pin through an inverter (IC3). When either ACIA is selected, A23 must be high, which pulls $\overline{VPA}$ low and initiates the synchronous data transfer. After the transfer is complete, the processor automatically resumes asynchronous operation. This scheme limits addresses using A23 to synchronous devices only.

Interrupt Handling

The 68000 has two modes of interrupt processing: normal and alternate. In the normal mode, the processor responds to an interrupt request by placing the level number of the interrupt on bits A1–A3 of the address bus and driving the function code lines FC0–FC2 high. The interrupting device then must place a vector number on the data bus and pull $\overline{DTACK}$ low to signal the 68000 that the vector number is available. The 68000 uses this vector number to acquire the service-routine address from the vector table in low memory. In the alternate mode, known as auto-vectoring, the processor again places the level number of the interrupt on A1–A3 and drives FC0–FC2 high. Then, instead of placing a vector number on the data bus, the interrupting device pulls the $\overline{VPA}$ processor pin low. This causes the processor to acquire the service-routine address from the position in the vector table that corresponds to the interrupt level. Therefore, in auto-vector mode, there are seven interrupt vectors, each of which is associated with one level of interrupt.

Address	Device
$000000–$000FFF	ROM
$001000–$001FFF	RAM
$A00000–$A00002	Terminal
$C00000–$C00002	Serial Port

Table 2: *The address map.*

The VU68K uses the auto-vector mode for interrupts. When the 68000 responds to an interrupt, all address bus lines except A1–A3 are driven high. Since A23 is connected to $\overline{VPA}$ through an inverter to support synchronous transfers, the processor is forced into the auto-vector mode for all interrupts.

The VU68K acknowledges only two levels of interrupt: level 1 for keyboard serial-port interrupts and level 2 for communication serial-port interrupts. Interrupts are signaled by lines $\overline{IPL0}$ and $\overline{IPL1}$. $\overline{IPL2}$ is always maintained high. This interrupt structure enables the keyboard and communication serial ports to be used simultaneously.

Address Decoding

The VU68K's memory is divided into 2K-word by 16-bit memory blocks, with a total potential of 32K words. The reason for this arrangement is that the VU68K uses only 11 of the 68000's address bits for word selection and another 4 address bits for memory-block selection. The 11 bits select one of 2K words, while the other 4 bits select one of sixteen 2K-word blocks. Since each block contains 2K words, the total memory capacity is 2K by 16 or 32K words.

The word-selection bits, A1–A11, go directly to each of the ROM and RAM ICs. The block-selection bits, A12–A15, go to a 4-to-16 decoder (IC5). The decoder, through the IC6 OR gates, pulls the output enable ($\overline{OE}$) lines of the selected 2K-word block low to activate that block. Since the data bus on the 68000 is 16 bits wide and the memory ICs used in the VU68K are each 8 bits wide, we had to use two memory ICs for each 16-bit-wide memory block. One of the memory ICs is assigned to the high-order 8 bits of the data bus, while the other is assigned to the low-order 8 bits.

The VU68K uses the 68000's $\overline{UDS}$ and $\overline{LDS}$ lines to control the method by which data is transferred to or from memory. If both lines are low, then data transfers occur in 16-bit words. If only one of the lines is low, then transfers occur one byte at a time. Note that two memory chips are active for a 16-bit transfer, while only one chip is selected for an 8-bit transfer. To select the correct device for a memory transfer, the output of the decoder is combined through an OR gate with the $\overline{UDS}$ and $\overline{LDS}$ signal to generate four signals—RAME, RAMO, ROME, ROMO—for selecting the correct memory device. The address map is shown in table 2.

To add memory or other asynchronous devices to the VU68K, all we have to do is select an address in the lower 64K bytes of memory. The high-order 4 bits of this new address cause one pin of the address decoder to be driven low. The output from this pin can then be used with additional decoding logic for selecting the new device. Adding a new asynchronous peripheral is even easier. Just select an address that sets address bit A23 and is not used by any other device. Addresses that set bit A23 are in the range $800000–$FFFFFF.

Let's assume that we wish to add another 2K-word block. We'll place two 6116 static RAM chips, 2K by 8 bits each, at addresses $002000–$002FFF. From the address map in

table 2 we can see that these locations are not currently used. The new RAM select line will be pin 3 of the address decoder output. Two additional OR gates will be required to ensure that the device assigned to the upper and lower portion of the bus is addressed only when dictated by the appropriate setting of $\overline{UDS}$ and $\overline{LDS}$. The address map in table 3 represents the new configuration.

If we want to add a new synchronous device, the address extension is even simpler. Let's assume that we wish to add another serial port. The Motorola 6850 we used earlier is a serial port chip that requires three chip selects, two in the high state and one in the low state. If we tie one of these to a high state and then use the other two select lines, we can safely use addresses $900000–$900002 for the device's control/status and data registers, respectively. This address avoids conflict with the other two synchronous devices and allows us to drive the device select lines without additional logic. The high select line will be address line A20, and the low select line will be address line A21. Table 4 is an address map that includes the new memory and serial port.

To enhance the interrupt structure so that not all interrupts are handled in the auto-vector mode, make sure that the synchronous mode-select line is still driven high when the terminal and serial ports are addressed. Discrete logic can easily remedy this problem.

Communications Interface

The basis of the VU68K's communication facility is the ACIA. This 8-bit device communicates synchronously on the data bus and asynchronously on the external RS-232C line. Both ACIA devices in the VU68K are initialized by the monitor software to handle full-duplex RS-232C lines with two stop bits and no parity. You can change these characteristics by moving new values to the terminal and serial port control registers at locations $A00000 and $C00000, respectively. The values to be loaded there can be determined from the Motorola 6850 data sheet.

Address	Device
$000000–$000FFF	ROM
$001000–$001FFF	RAM
$002000–$002FFF	New RAM
$A00000–$A00002	Terminal
$C00000–$C00002	Serial Port

Table 3: *The extended address map.*

Address	Device
$000000–$000FFF	ROM
$001000–$001FFF	RAM
$002000–$002FFF	New RAM
$900000–$900002	New Serial Port
$A00000–$A00002	Terminal
$C00000–$C00002	Serial Port

Table 4: *The extended address map with a serial port added.*

Vector Address	Interrupt
$1000	User Trap-vector B
$1004	User Trap-vector C
$1008	User Trap-vector D
$100C	User Trap-vector E
$1010	User Trap-vector F
$1014	User Interrupt-vector 1
$1018	User Interrupt-vector 2
$101C	User Interrupt-vector 3
$1020	User Interrupt-vector 4
$1024	User Interrupt-vector 5
$1028	Auto-vector Level 3
$102C	Auto-vector Level 4
$1030	Auto-vector Level 5
$1034	Auto-vector Level 6
$1038	Auto-vector Level 7

Table 5: *User interrupt-vectors.*

We added two line conditioners after the ACIAs to provide RS-232C logic levels to the external lines. We used 1488 and 1489 quad line-drivers/receivers (IC14, IC15) for this purpose. A 14411 bit-rate frequency generator (IC13) and a 1.8432-MHz crystal tell the ACIAs at what data rate they may transmit and receive on the RS-232C lines. Data rates are switch selectable through a dual in-line package (DIP) switch and may be selected independently for either the terminal or communication serial port. The allowable data rates are 300, 1200, 2400, and 9600 bps. You can use other data rates by connecting the DIP switch to the proper pins on the 14411. All interrupts generated by the terminal or communication serial port are handled by the interrupt-service routines in the monitor, which we will describe later.

Miscellaneous Support

The only devices we haven't yet covered are the reset and clock circuits. The reset circuit is simply a debounced switch that pulls both the $\overline{RESET}$ and $\overline{HALT}$ pins of the processor low. These pins must be held low for at least 100 milliseconds for the reset operation to function correctly. In the VU68K, we used a push-on/push-off switch to do this. You can use a conventional momentary contact switch if you ensure that the switch is off for at least 100 milliseconds.

The clock circuit is an oscillator circuit driven by a 5-MHz crystal. The resulting signal, CLK, drives both the processor clock and the counter used in asynchronous accesses.

A Monitor Program

To support the hardware of the VU68K, we developed a comprehensive monitor program called VUBUG. VUBUG provides a set of program-development support services that includes I/O (input/output) buffering, program-development commands, trap handlers, and error-handling utilities.

VUBUG provides buffered I/O for both the terminal and communication serial port. Separate interrupt-handling routines for these devices located at level 1 and level 2 of the interrupt structure, respectively, provide a complete set of facilities for implementing concurrent buffered I/O. When a port generates an interrupt, VUBUG causes the processor to read the port and place the data in a buffer that can store 16 bytes. A trap instruction, which we will discuss later, retrieves the data. Interrupts from the terminal keyboard cause the character read to be echoed immediately as well as buffered.

The VUBUG interrupt structure includes five auto-vectors, the first five user interrupt-vectors, and five user trap-vectors, which are loaded by the monitor to point to the locations shown in table 5. At each of these

locations, you may assemble a branch instruction to an exception handler that you provide. The 4 bytes set aside for each vector allows for a branch with a 16-bit displacement. Be careful to terminate these exception handlers with an RTE instruction to ensure proper return from the exception handler.

VUBUG supports a set of commands that we've selected to support the goal of simplicity. Table 6 summarizes the available commands and subcommands. Perhaps the most powerful command for program development is a combination of the Trace (t+) and Breakpoint (b+) commands. Trace is an instruction-by-instruction trace of the value of the user program counter; it shows the

Command	Action
m<cr>	Start memory mode
m xxxx	Start memory mode at xxxx
.xxxx	Set pointer to xxxx
=xx	Store value xx at address in pointer
,xx	Increment pointer and store xx
+	Increment pointer and display value
–	Decrement pointer and display value
l<cr>	Start program load
l xxxx	Start program load and offset each block by xxxx bytes
d<cr>	Display the next 80 bytes from memory pointer
d xxxx<cr>	Display 80 bytes starting at address xxxx
d xxxx,yyyy	Display all data between locations xxxx and yyyy inclusive
t+	Start trace
t–	Stop trace
s+	Start single step
s–	Stop single step
e	Start terminal emulator mode
g xxxx	Start program at address xxxx
g<cr>	Start user program from address in user PC
<cr>	Same as g<cr>
b+xxxx	Insert a breakpoint at address xxxx
b–xxxx	Remove a breakpoint at address xxxx
b<cr>	Show all breakpoints
b#	Remove all breakpoints
r<cr>	Start register mode
r xx	Start register mode at register xx where xx is: SR/sr status register PC/pc program counter D0/d0 –D7/d7 data registers A0/a0 – A7/a7 address registers
.xx	Set register pointer to register xx
=xxxxxxxx	Store value in register at pointer for SR value is xxxx
<cr>	Print values in all registers
px yyyy	Associate with px the program starting at address yyyy; where x is 1, 2, or 3
px<cr>	Execute command px where x is 1, 2, or 3
c xxxx=yyyy,zzzz	Copy from location yyyy through location zzzz to locations starting at xxxx and increasing

Table 6: *Command summary.*

Trap	Function	Place Argument in Register	Return Argument is Placed in Register
0	Exit	None	None
1	Get byte	None	D0
2	Get word	None	D0
3	Get long	None	D0
4	Write byte	D0	None
5	Write word	D0	None
6	Write long	D0	None
7	Get character	None	D0
8	Write string	A0	None
9	Write character	D0	None
A	Write cr-lf	None	None

Table 7: *VUBUG traps.*

program's execution path. Breakpoint stops the program at the address you select. With these two commands, a user may see the instructions leading to a breakpoint and then use the other commands available in VUBUG to examine registers and memory to determine why that path was taken or to modify the path that will be taken after the breakpoint.

The VUBUG monitor provides 11 traps for servicing user-program requests. These traps are called simply by executing the appropriate trap instruction with the appropriate argument, as shown in table 7. In addition to these trap handlers, you can use traps B through F with the vectors at locations shown in table 5.

Error handling, or exception processing, for processor-detected errors, is also provided by VUBUG. Error handling involves intercepting the interrupt and reporting the error on the terminal. In addition, the register values are copied into the register save area and are accessible via the r command. The errors trapped include address/bus errors, illegal instruction errors, privilege violations, and a class of generic errors that share a single error handler. The errors in this class are zero divide, CHK and TRAPV, and spurious interrupts.

Rounding Out

For a complete development system, you'll need an RS-232C terminal, a power supply that provides +5 volts at 0.5 amperes, and +12 and −12 volts at 0.1 amperes, and a host computer.

Development software on the host should include a disassembler, high-level language processors, cross-reference builders, machine simulators, and so forth. The only additional software required is a simple program to send the object program to the serial port for loading. Only your imagination and the capabilities of the host limit its use.■

This system is currently running at Vanderbilt University and is proving to be an easy method for designing and testing software to run on the VU68K. Readers who desire more information on the VU68K and VUBUG can, for a nominal fee, order a copy of Vanderbilt University Computer Science Technical Report CS-83-01 from the following address:

POB 1679
Station B
Vanderbilt University
Nashville, TN 37235

The authors would like to thank the Computer Science and Electrical Engineering departments of Vanderbilt University for their support in VU68K system design and realization. Special thanks to Motorola Corporation for its assistance in the hardware design.

Edward M. Carter teaches computer science at the U.S. Air Force Academy. This work was completed while he was doing graduate work at Vanderbilt University in Nashville. He has a B.S. from the U.S. Air Force Academy, an M.S. from U.C.L.A., and a Ph.D. from Vanderbilt. He can be reached at USAFA/DFCS, USAFA CO 80840.

A. B. Bonds teaches electrical engineering at Vanderbilt. He holds a Ph.D. in electrical engineering and is currently studying models of information processing in mammalian visual systems. He also likes "to mess with antique autos." He can be reached at Vanderbilt University, POB 1824B, Nashville, TN 37235.